Homework 5: Q3

Name: Waiwai Kim, waiwaiki

1 Part (a): Algorithm Idea

The algorithm presented in the this week recitation notes is a simple brute-force search that iterate over all n^3 triples of distinct vertices. The algorithm idea and details are presented in the details in the notes. The algorithm runs in $O(n^3)$ time assuming that we can check whether $\{u, v, w\}$ is a triangle or not in a constant time of $O(1)$. This is true because we can answer if (u, v) is an edge give a pair of u, v of vertices. In an adjadency list indexed at u, it takes a constant time to check if v is in the list. Checking three edges also takes a constant time. Thus, check-if-triag can be implemented in $O(1)$.

2 Part (b), subpart-1: Algorithm Idea

There is no need to check all possible triples. We only need to check vertex triples when we already know two edges exist to form a triangle exists. For example, if we already know if (A, B) and (A, C) edges exist, we only need to check if (B, C) exists. Assuming that we are given an adjacency list, pick a vertex A. We already know that any two vertices in the list with key A, say B and C, have edges from A. These two edges form two sides of a triangle. We are left with checking if there is an edge in (B, C).

3 Part (b), subpart-1: Algorithm Details

Algorithm 1 Algorithm1

```
1: procedure ALGORIGHM-1(Inputs)
2:     for key, value in graph.items(): do
3:         enumerate a pair (vertex[i], vertex[i+1])
4:         check if an exist exists between vertex[i] and vertex[i]
5:         if Yes: then
6:             add to the triangle list
7:         else
8:             pass to the next combination of two vertices
9:         end if
10:    end for
        return triangle list
11: end procedure
```

4 iPart (b), subpart-1: Proof of Correctness Idea

5 Part (b), subpart-1: Runtime Analysis

Here we prove that the runtime of the aforementioned algorithm is Equation 1. The equation 1 can be shown in two different form: in Equation 3 and 2. As the algorithm details above shows, the algorithm enumerates all n vertices. Given a vertex, we need to check $C(k,2)$ pairs of vertices. Here $C(,)$ stands for combination and k stands for the degree of the given vertex. The upper bound of $C(k,2)$ is $C(\Delta,2)$, which can be expressed as $O(\Delta^2)$ as shown in Equation 4.This shows that the runtime is indeed bound by Equation 2. However, $C(\Delta,2)$ is constrained by the actual number of existing edges or m. For example if $m = 10$ while $\Delta = 4$, we don't need to check all 16 possible combinations because there are only 10. Thus, the runtime takes the smaller of m and Δ^2.

$$O(n \cdot min(m, \Delta^2)) \tag{1}$$

$$O(n \cdot \Delta^2) \;\; when \;\; m > \Delta^2 \tag{2}$$

$$O(n \cdot m) \;\; when \;\; m < \Delta^2 \tag{3}$$

$$C(\Delta,2) = \frac{\Delta(\Delta-1)}{2} = O(\Delta^2) \tag{4}$$

6 Part (b), subpart-2: Algorithm Idea

The algorithm presented above still has redundancy. It finds each triangle three times. For example, let's talk about triangle $(2,6,7)$ in Figure 1. Instead counting the triangle three times starting from each node, we can count once starting from 6, which has the smallest degree amount its neighbors. $deg(2) = 5$ and $deg(7) = 3$, while $deg(6)$=2. This is possible because there are always "roads leading to nodes with high degree" so to speak.

7 Part (b), subpart-2: Runtime Analysis

How do we tell which vertex has a small degree and which vertex has a large degree? We can use $min(m, \Delta^2)$ notion. This implies if we have vertex k such that $deg(k) < \sqrt{m}$, the algorithm iterates less. On the other hand, high degree vertex has $deg(k)$ of at least $\sqrt{m}$ or greater than $\sqrt{m}$. We separate our computation into two parts of 1) low degree vertices and 2) high degree vertices.

The upper bound of low degree vetices is based the fact that the maximum degree of small degree vertices is $\sqrt{m}$. Also because the total number of degree is $2m$ (m edges going both ways), the number of vertices w/ $deg(\sqrt{m})$ is limited at $2\sqrt{m}$. Thus, the sum of computation done by small vertices is equal to $O(\sqrt{m} \times \sqrt{m}^2) = O(m^{\frac{3}{2}})$.

On the other hand, the upper bound of computation by high degree vertices is constrained by the maximum number of such vertcies. We know that there at most $2\sqrt{m}$ large vertices. Enumerating all possible triples of $2\sqrt{m}$ large vertices leads to $O(m^{\frac{3}{2}})$.

Combining the computation done by small and large vertices show that the total runtime of the algorithm is $O(m^{\frac{3}{2}})$.

Figure 1: Find Triangles - Example Graph